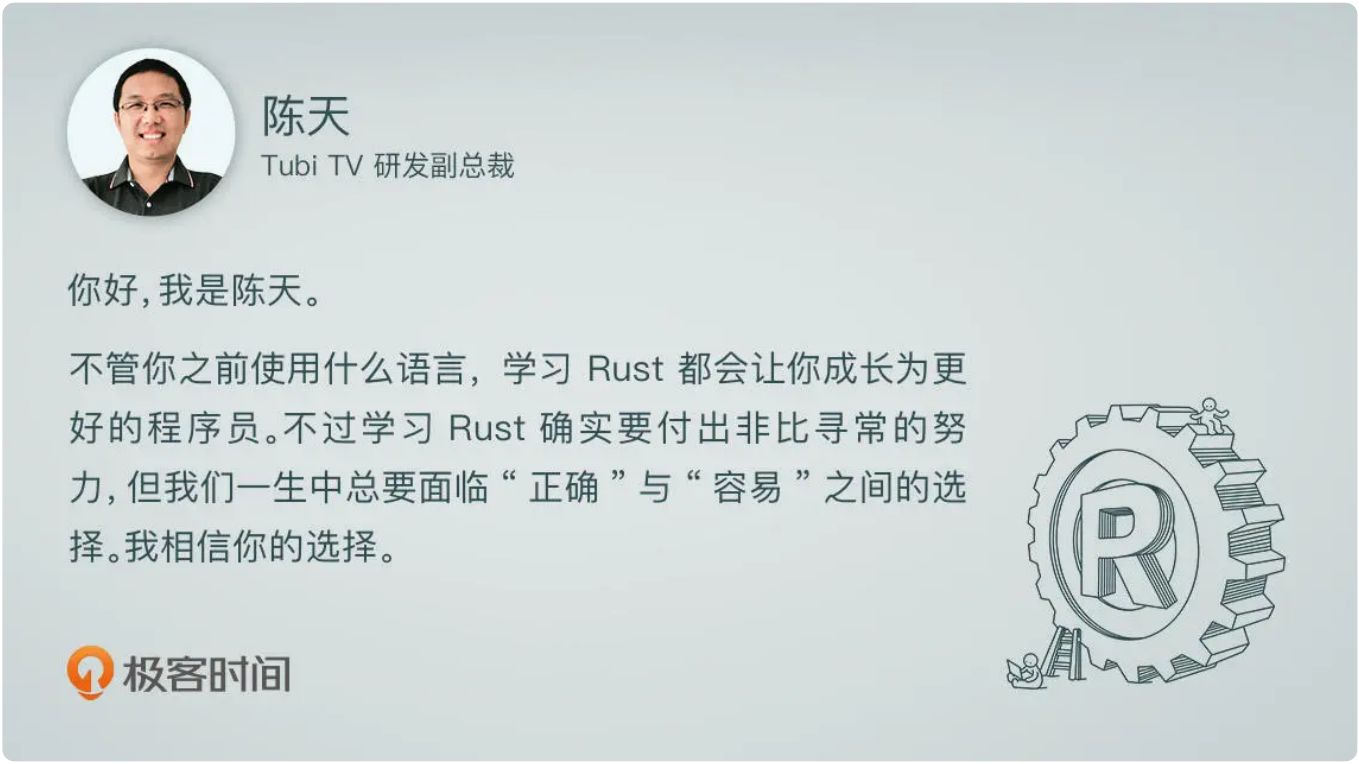
结束语｜永续之原：Rust学习，如何持续精进？

2021-12-22 陈天

课程介绍 >



讲述：陈天
时长 13:13　大小 12.12M

你好，我是陈天。

首先，恭喜你完成了这门课程！

六月底，我确定了自己会在极客时间上出这个 Rust 的专栏。

其实以前我对这样子的付费课程不是太感冒，因为自己随性惯了，写公众号自由洒脱，想写就写，想停就停，一个主题写腻了还可以毫无理由地切换到另一个主题上。但一旦写付费的专栏签下合同，就意味着品味、质量、内容以及更新的速度都不能随心所欲，得按人家的要求来。

最要命的是更新的速度——我没有专职做过文字工作者，想来和代码工作者性质类似，一些开创性工作的开始特别需要灵感，非常依赖妙手偶得的那个契机。这种不稳定的输出模式，遇到进度的压力，就很折磨人。所以之前很多机会我都婉拒了。

但这次思来想去，我还是接下了 Rust 第一课这个挑战。

大部分原因是我越来越喜爱 Rust 这门语言，想让更多的人也能爱上它，于是之前在公众号和 B 站上，也做了不少输出。但这样的输出，左一块右一块的，没有一个完整的体系，所以有这样一个机会，来构建出我个人总结的 Rust 学习体系，也许对大家的学习会有很大的帮助。

另外一部分原因也是出于我的私心。自从 2016 年《途客圈创业记》出版后，我就没有正式出版过东西，很多口头答应甚至签下合同的选题，也都因为各种原因被我终止或者搁置了。我特别想知道，自己究竟是否还能拿起笔写下严肃的可以流传更广、持续更久的文字。

可是——**介绍一门语言的文字可以有持久的生命力么？**

你一定有这个疑问。

撰写介绍一门编程语言的文字，却想让它拥有持久的生命力，这听上去似乎是痴人说梦。现代编程语言的进化速度相比二十年前，可谓是一日千里。就拿 Rust 来说，稳定的六周一个版本，三年一个版次，别说是拥有若干年的生命力了，就算是专栏连载的几个月，都会过去两三个版本，也就意味着有不少新鲜的东西被加入到语言中。

不过好在 Rust 极其注重向后兼容，也就意味着我现在介绍的代码，只要是 Rust 语言或者标准库中稳定的内容，若干年后（应该）还是可以有效的。Rust 这种不停迭代却一直保持向后兼容的做法，让它相对于其它语言在教学上有一些优势，所以，撰写介绍 Rust 的文字，生命力会更加持久一些。

当然这还远远不够。让介绍一门编程语言的文字更持久的方式就是，**从本原出发，帮助大家理解语言表层概念背后的思想或者机理，**这也是这个专栏最核心的设计思路。

通用型计算机诞生后差不多七十年了，当时的冯诺依曼结构依然有效；从 C 语言诞生到现在也有快五十年了，编程语言处理内存的方式还是堆和栈，常用的算法和数据结构也还是那些。虽然编程语言在不断进化，但解决问题的主要手段还都是差不多的。

比如说，引用计数，你如果在之前学习的任何一门语言中弄明白了它的思路，那么理解 Rust 下的 Rc/Arc 也不在话下。所以，只要我们把基础知识夯实，很多看似难懂的问题，只不过是在同样本质上套了让人迷惑的外衣而已。

那么如何拨开迷雾抵达事物的本原呢？我的方法有两个：**一曰问，二曰切**。对，就是中医“望闻问切”后两个字。

问就是刨根追底，根据已有的认知，发出直击要害的疑问，这样才能为后续的探索（切）叩开大门。比如你知道引用计数通行的实现方法，也知道 Rust 的单一所有权机制把堆内存的生命周期和栈内存绑定在一起，栈在值在，栈亡值亡。

那么你稍微思考一下就会产生疑问：Rc/Arc 又是怎么打破单一所有权机制，做到让堆上的内存跳脱了栈上内存的限制呢？问了这个问题，你就有机会往下“切”。

“切”是什么呢，就是深入查看源代码，顺着脉络找出问题的答案。初学者往往不看标准库的源码，实际上，看源代码是最能帮助你成长的。无论是学习一门语言，还是学习 Linux 内核或者别的什么，源码都是第一手资料。别人的分析讲得再好，也是嚼过的饭，受限于他的理解能力和表达能力，这口嚼过的饭还真不一定比你自己亲自上嘴更好下咽。

比如想知道上面 Rc/Arc 的问题，自然要看 Rc::new 的源码实现：

复制代码

```
pub fn new(value: T) -> Rc<T> {
    // There is an implicit weak pointer owned by all the strong
    // pointers, which ensures that the weak destructor never frees
    // the allocation while the strong destructor is running, even
    // if the weak pointer is stored inside the strong one.
    Self::from_inner(
        Box::leak(box RcBox { strong: Cell::new(1), weak: Cell::new(1), value
    )
}
```

不看不知道，一看吓一跳。可疑的 Box::leak 出现在我们眼前。这个 Box::leak 又是干什么的呢？顺着这个线索追溯下去，我们发现了一个宝贵的金矿（你可以回顾生命周期的那一讲）。

思

在追溯本原的基础上，我们还要学会分析问题和解决问题的正确方法。我觉得编程语言的学习不应该只局限于学习语法本身，更应该在这个过程中，不断提升自己学习知识和处理问题的能力。

如果你还记得 HashMap 那一讲，我们先是宏观介绍解决哈希冲突的主要思路，它是构建哈希表的核心算法；然后使用 transmute 来了解 Rust HashMap 的组织结构，通过 gdb 查看内存布局，再结合代码去找到 HashMap 构建和扩容的具体思路。

这样**一层层剥茧抽丝，边学习，边探索，边总结**，最终我们得到了对 Rust 哈希表非常扎实的掌握。这种掌握程度，哪怕你十年都不碰 Rust，十年后有人问你 Rust 的哈希表怎么工作的，你也能回答个八九不离十。

我希望你能够掌握这种学习的方式，这是终生受益的方式。2006 年，我在 Juniper 工作时，用类似的方式，把 ScreenOS 系统的数据平面的处理流程总结出来了，到现在很多细节我记忆犹新。

很多时候面试一些同学，详细询问他们三五年前设计和实现过的一些项目时，他们会答不上来，经常给出“这个项目太久了，我记不太清楚”这样的答复，让我觉得好奇怪。对我而言，只要是做过的项目、阅读过的代码，不管多久，都能回忆起很多细节，就好像它们是自己的一部分一样。

尽管快有 20 年没有碰，我还记得第一份工作中 OSPFv2 和 IGMPv3 协议的部分细节，知道 netlink 如何工作，也对 Linux VMM 管理的流程有一个基本印象。现在想来，可能就是我掌握了正确的学习方法而已。

所以，在这门介绍语言的课程中，我还夹带了很多方法论相关的私货，它们大多散落在文章的各个角落，除了刚刚谈到的分析问题 / 解决问题的方法外，还有阅读代码的方法、架构设计的方法、撰写和迭代接口的方法、撰写测试的方法、代码重构的方法等等。希望这

些私货能够让你产生共鸣，结合你自己在职业生涯中总结出来的方法，更好地服务于你的学习和工作。

读

在撰写这个专栏的过程中，我参考了不少书籍。比如《Programming Rust》、《Designing Data-intensive Applications》以及《Fundamentals of Software Architecture》。可惜 Jon Gjengset 的《Rust for Rustaceans》姗姗来迟，否则这个专栏的水准可以更上一个台阶。

我们做软件开发的，似乎到了一定年纪就不怎么阅读，这样不好。毕加索说：“good artists copy; great artists steal.”当你从一个人身上学习时，你在模仿；当你从一大群人身上学习时，你自己就慢慢融会贯通，成为大师。

所以，不要指望学了这门 Rust 第一课，就大功告成，**这门课仅仅是一个把你接引至 Rust 世界的敲门砖，接下来你还要进一步从各个方面学习和夯实更多的知识。**

就像我回答一个读者的问题所说的：很多时候，我们缺乏的不是对 Rust 知识的理解，更多是对软件开发更广阔知识的理解。所以，不要拘泥于 Rust 本身，对你自己感兴趣的，以及你未来会涉猎的场景广泛阅读、深度思考。

行

伴随着学习，阅读，思考，我们还要广泛地实践。不要一有问题就求助，想想看，自己能不能构造足够简单的代码来帮助解决问题。

比如有人问：HTTP/2 是怎么工作的？这样的问题，你除了可以看 RFC，阅读别人总结的经验，还可以动动手，几行代码就可以获得很多信息。比如：

复制代码

```
use tracing::info;
use tracing_subscriber::EnvFilter;

fn main() {
    tracing_subscriber::fmt::fmt()
        .with_env_filter(EnvFilter::from_default_env())
        .init();

```

```
    let url = "<https://www.rust-lang.org/>";

    let _body = reqwest::blocking::get(url).unwrap().text().unwrap();
    info!("Fetching url: {}", url);
}
```

这段代码相信你肯定能写得出来，但你是否尝试过 RUST_LOG=debug 甚至 RUST_LOG=trace 来看看输出的日志呢？又有没有尝试着顺着日志的脉络，去分析涉及的库呢？

下面是这几行代码 RUST_LOG=debug 的输出，可以让你看到 HTTP/2 基本的运作方式，我建议你试试 RUST_LOG=trace（内容太多就不贴了），如果你能搞清楚输出的信息，那么 Rust 下用 hyper 处理 HTTP/2 的主流程你就比较明白了。

复制代码

```
❯ RUST_LOG=debug cargo run --quiet
2021-12-12T21:28:00.612897Z DEBUG reqwest::connect: starting new connection: <
2021-12-12T21:28:00.613124Z DEBUG hyper::client::connect::dns: resolving host=
2021-12-12T21:28:00.629392Z DEBUG hyper::client::connect::http: connecting to
2021-12-12T21:28:00.641156Z DEBUG hyper::client::connect::http: connected to 1
2021-12-12T21:28:00.641346Z DEBUG rustls::client::hs: No cached session for Dn
2021-12-12T21:28:00.641683Z DEBUG rustls::client::hs: Not resuming any session
2021-12-12T21:28:00.656251Z DEBUG rustls::client::hs: Using ciphersuite Tls13(
2021-12-12T21:28:00.656754Z DEBUG rustls::client::tls13: Not resuming
2021-12-12T21:28:00.657046Z DEBUG rustls::client::tls13: TLS1.3 encrypted exte
2021-12-12T21:28:00.657151Z DEBUG rustls::client::hs: ALPN protocol is Some(b"
2021-12-12T21:28:00.658435Z DEBUG h2::client: binding client connection
2021-12-12T21:28:00.658526Z DEBUG h2::client: client connection bound
2021-12-12T21:28:00.658602Z DEBUG h2::codec::framed_write: send frame=Settings
2021-12-12T21:28:00.659062Z DEBUG Connection{peer=Client}: h2::codec::framed_w
2021-12-12T21:28:00.659327Z DEBUG hyper::client::pool: pooling idle connection
2021-12-12T21:28:00.659674Z DEBUG Connection{peer=Client}: h2::codec::framed_w
2021-12-12T21:28:00.672087Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.672173Z DEBUG Connection{peer=Client}: h2::codec::framed_w
2021-12-12T21:28:00.672244Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.672308Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.672351Z DEBUG Connection{peer=Client}: h2::proto::settings
2021-12-12T21:28:00.956751Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.956921Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.957015Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.957079Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:00.957316Z DEBUG reqwest::async_impl::client: response '200 O
2021-12-12T21:28:01.018665Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:01.018885Z DEBUG Connection{peer=Client}: h2::codec::framed_r
2021-12-12T21:28:01.020158Z  INFO http2: Fetching url: <https://www.rust-lang.
```

所以，很多时候，知识就在我们身边，我们写一写代码就能获取。

在这个过程中，你自己思考之后撰写的探索性的代码、你分析输出过程中付出的思考和深度的阅读，以及最后在梳理过程中进行的总结，都会让知识牢牢变成你自己的。

最后我们聊一聊写代码这个事。

学习任何语言，最重要的步骤都是用学到的知识，解决实际的问题。Rust 能不能胜任你需要完成的各种任务？大概率能。但你能不能用 Rust 来完成这些任务？不一定。每个十指俱全的人都能学习弹钢琴，但不是每个学弹钢琴的人都能达到十级的水平。这其中现实和理想间巨大的鸿沟就是“刻意练习”。

想要成为 Rust 专家，想让 Rust 成为你职业生涯中的一项重要技能，刻意练习必不可少，需要不断地撰写代码。的确，Rust 的所有权和生命周期学习和使用起来让人难于理解，所有权、生命周期，跟类型系统（包括泛型、trait），以及异步开发结合起来，更是障碍重重，但通过不断学习和不断练习，你一定会发现，它们不过是你的一段伟大旅程中越过的一个小山丘而已。

最后的最后，估计很多同学都是在艰难斗争、默默学习，在专栏要结束的今天，我也非常希望能听到你的声音，听听你学习这个专栏的感受和收获，见到你的身影。

感谢你选择我的 Rust 第一课。感谢你陪我们一路走到这里。接下来，就看你的了。

> “Go where you must go, and hope!” — Gandalf

分享给好友，一起充电升级

赞 11　提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　用户故事 | 语言不仅是工具，还是思维方式

更多课程推荐

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。